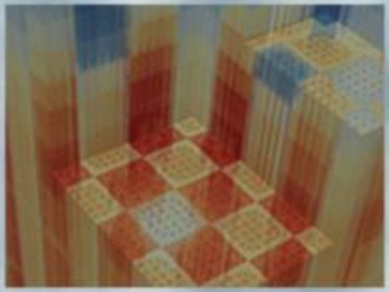
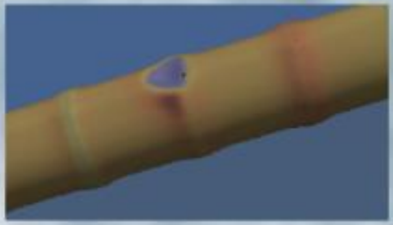
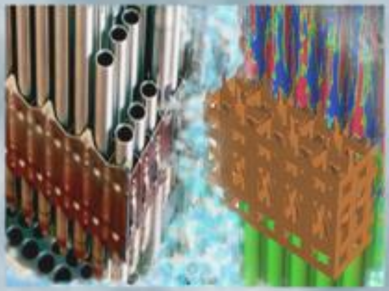
CASL-U-2017-1307-000

Milestone Report: Non-Source VERA Release

Damien Lebrun-Grandié
Mark Baird
Kevin Clarno
Tony Walsh
Mitchell Young

3 April 2017

REVISION LOG

Revision	Date	Affected Pages	Revision Description
0	3 April 2017	All	Initial Public Release - Unlimited

Document pages that are:

Export Controlled ___________________None___________________________________

IP/Proprietary/NDA Controlled_______None____________________________________

Sensitive Controlled__________________None___________________________________

Requested Distribution:

To:

Copy:

EXECUTIVE SUMMARY

This report documents the completion of milestone: L3:PHI.INF.P14.02 to develop a "Non-Source Release of VERA." Three options were investigated with two being complete (Docker image and InSPECT cluster) and one is still in progress (AmazonCloud). The VERA-3.6 code was made available on InSPECT and several users have access to use that version via RSICC approval. A Docker image was created for the VERA-3.6 code along with all of the required data. This was independently tested on LEZA and U233 through an ssh-tunnel that ensures the user does not have inappropriate permissions to access system data as root. Finally, approvals are in-progress to get VERA on the ORNL account for the AmazonCloud (because it is export controlled). In the meantime, the CTF Docker image was ported to the Cloud and can be used.

This report documents the means to generate and use the Docker image on a cluster as well as a clear description of the other efforts. The milestone description was to: "Based on the source-release of VERA in PoR14, create a means for users to execute VERA without source code access. This might be through Dockers, the Inspect cluster, Amazon Cloud, or a different approach." While the execution plan contained, "Create a means for users to execute VERA without source code access and document work." The completion criteria, which has clearly been met, includes "A non-source release based on the PoR14 version of VERA is available to users who do not have approval for the source code."

CONTENTS

ACRONYMS

AWS	Amazon Web Services
CASL	Consortium for Advanced Simulation of Light Water Reactors
DOE	US Department of Energy
LWR	light water reactor
ORNL	Oak Ridge National Laboratory
RSICC	Radiation Safety Information Computational Center
VERA	Virtual Environment for Reactor Applications

1. INTRODUCTION

This report details the process of creating and using non-source VERA distributions. Two packaging approaches were used: using the Docker containerization system, and using the InSPECT cluster at ORNL.

The Docker approach is a fully produces a fully portable image, which may be used to run VERA within a container on any system running Docker. Section 2 outlines the steps necessary to create the Docker image. Section 3 describes how to deploy the VERA container to a system running Docker.

The InSPECT approach, described in Section 5 avoids the virtualization layer needed to use Docker by mirroring the software environment between two systems; one system contains the souce code needed to build VERA, while the other system lacks the source, but preserves the libraries and other dependencies necessary for VERA to run.

Section 4 discusses the potential for using a containerized VERA distribution on the AWS GovCloud system.

2. CREATE DOCKER IMAGE FOR VERA RELEASE X.Y.Z

This section describes how to produce a Docker image for VERA from a base `vera_tpls:A.B.C` image with the Dev Env and TPLs.

2.1. GET THE VERA SOURCE CODE WITH PROPER TAGGED VERSION

Obtain the VERA source base directory with the VERA components to install. Several options here:

- Download the appropriate VERA tarball hosted on the fissile machines in `/projects/vera/tarballs/` and extract it:

```
[host]$ scp <fissile machine>:/projects/vera/tarballs/vera-X.Y.Z-Source.tar.gz .
[host]$ tar -xzf vera-X.Y.Z-Source.tar.gz
```

- Or, check out the tagged version with `git-dist`:

```
[host]$ cd $VERA_BASE_DIR
[host]$ git-dist checkout vera-X.Y.Z # checkout specific tag
```

2.2. SET UP A CONTAINER TO BUILD VERA

a. Create a Compose file to configure the container such as:

```
version: '2'
services:
  release_build:
    image: ${VERA_TPLS_DOCKER_IMAGE}
    environment:
      - VERA_DIR=${CONTAINER_VERA_DIR}
      - VERA_INSTALL_DIR=${CONTAINER_VERA_INSTALL_DIR}
      - VERA_BUILD_DIR=${CONTAINER_VERA_BUILD_DIR}
    volumes:
      - ${HOST_VERA_DIR}:${CONTAINER_VERA_DIR}
      - build_dir:${CONTAINER_VERA_BUILD_DIR}
    command: tail -f /dev/null
volumes:
  build_dir:
    driver: local
```

b. Provide a ".env" file to set the values of the environment variables referenced in the Compose file.

environment variable	description
`VERA_TPLS_DOCKER_IMAGE`	VERA base image with the Development Environment and TPLs (e.g. `dalg24/vera_tpls:A.B.C` hosted on Docker Hub)
`HOST_VERA_DIR`	path to the VERA base source directory on the host (e.g. `./vera-X.Y.Z-Source`)
`CONTAINER_VERA_DIR`	mount point for the VERA base source directory within the Docker container. (e.g `/scratch/vera-X.Y.Z-Source`)
`CONTAINER_VERA_INSTALL_DIR`	install directory within the container (e.g. `/tools/vera_installs/YYYY-MM-DD`)
`CONTAINER_VERA_BUILD_DIR`	build directory within the container (e.g. `/scratch/vera_build` which comes with a `do-configure.sh` script)
`COMPOSE_PROJECT_NAME`	[optional] set the project name (value is used as a prefix when naming the container and the volume that is attached to it)

Note that values of the environment variables in the shell take precedence over those specified in the `.env` file.

One may use the `docker-compose config` command to print the configuration to the terminal.

c. Start the container in the background and leave it running:

```
[host]$ docker-compose up -d
```

The command will create the data volume that will serve as scratch space for the build directory if it does not already exists and mount it into the container.

One may list containers and volumes, by doing:

```
[host]$ docker-compose ps
          Name                      Command           State    Ports
---------------------------------------------------------------------
veraXYZ_release_build_1     tail -f /dev/null     Up
```

and

```
[host]$ docker volume ls
DRIVER              VOLUME NAME
local               veraXYZ_build_dir
```

respectively.

2.3. INSTALL VERA WITHIN THE CONTAINER

Launch an interactive bash session within the container. Configure, build, test, and install VERA as on would usually do:

```
[host]$ docker exec -it veraXYZ_release_build_1 bash
[container]# cd $VERA_BUILD_DIR
[container]# ./do-configure.sh
[container]# make -j16
[container]# ctest -j16
[container]# make install
[container]# exit
```

2.4. COMMIT THE NEW IMAGE AND TURN IT INTO A TARBALL ARCHIVE

Create a new image with the installed release version of VERA:

```
[host]$ docker commit veraXYZ_release_build_1 vera:X.Y.Z
```

The commited image will not include any data contained in volumes mounted inside the container. One may apply all necessary extra changes to the container's file before creating the commit.

Save the image to a `tar.gz` archive:

```
[host]$ docker save vera:X.Y.Z | gzip > vera-X.Y.Z-Docker.tar.gz
```

2.5. CLEANUP

Stop and remove the container that was created with `docker-compose up`:

```
[host]$ docker-compose down
```

Append `--volume` to the command to remove the volume that was attached to the container.

2.6. [APPENDIX] USE THE VERA RELEASE IMAGE

a. Get the Docker image with the VERA binaries. Archive `.tar.gz` files are available for download on the fissile machines in the `/projects/vera/tarballs/docker` directory.

b. Expand the VERA Docker archive file and load the image:

```
[host]$ gunzip -c vera-X.Y.Z-Docker.tar.gz | docker load
Loaded image: vera:X.Y.Z
```

Now list the images, it will show the VERA image that was loaded from the tarball:

```
[host]$ docker images
REPOSITORY          TAG            IMAGE ID        CREATED         SIZE
vera                X.Y.Z          c4a647acc44a    2 hours ago     3.48 GB
```

c. Run a container from the VERA release image and launch an interactive bash shell inside of it:

```
[host]$ docker run --rm -it -v $PWD:/workspace -w /workspace vera:X.Y.Z bash
[container]# source $VERA_INSTALL_DIR/load_env.sh
[container]# # etc.
```

3. DEPLOY DOCKER CONTAINERS FOR VERA USERS

This section describes how to deploy Docker containers with binary release of VERA from an image `vera:X.Y.Z` and give access to users.

3.1. EXTEND THE IMAGE FOR VERA RELEASE X.Y.Z

Install and setup a SSH server in the VERA container that users can use to connect from some remote host. Below is an example of Dockerfile that takes the user public key as an argument at build time and adds it to the authorized keys file for root.

```
FROM vera:X.Y.Z

RUN apt-get update && \
    apt-get install -y openssh-server && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* && \
    mkdir -p /var/run/sshd

RUN echo "export VERA_TPL_INSTALL_DIR=${VERA_TPL_INSTALL_DIR}" >> /etc/profile
RUN echo "export VERA_INSTALL_DIR=${VERA_INSTALL_DIR}" >> /etc/profile

ARG USER_SSH_PUBLIC_KEY
COPY ${USER_SSH_PUBLIC_KEY} /root/.ssh/authorized_keys

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Add `RUN` instructions as necessary to install any tool that may be missing in the VERA image (such as a text editor).

3.2. SET UP THE CONTAINER FOR DEPLOYMENT

It is recommended to use ‘docker-compose‘ to configure containers. Here is an exemple of Compose file to deploy VERA and allow a user to connect into the container via SSH:

```
version: '2'
services:
  release_deploy:
    build:
      dockerfile: Dockerfile
      context: .
      args:
        - USER_SSH_PUBLIC_KEY=${USER_SSH_PUBLIC_KEY}
    environment:
      - TERM=xterm
      - TZ=america/new_york
    ports:
      - ${USER_ALTERNATE_SSH_PORT}:22
```

Consider using a data volume to make the user workspace persistent (i.e. not deleted when container is removed).

It is a good idea to use `.env` file for each user and to save that environment somewhere. The table below summarizes the environment variable that should be defined:

environment variable	description
`USER_SSH_PUBLIC_KEY`	path to the user SSH public key (e.g. `./johnny_id_rsa.pub`) NOTE: must be inside the build's context, `../something` or `/something` will raise an error at buildtime
`USER_ALTERNATE_SSH_PORT`	host port the container's port 22 will be mapped to (e.g. 2222)
`COMPOSE_PROJECT_NAME`	[recommended] used as a prefix when naming the container to avoid collisions when running multiple instances (e.g. `johnny`)

Start the container with:

```
[host]$ docker-compose -p <user> up -d --build
```

3.3. PROVIDE THE USER WITH INSTRUCTIONS TO CONNECT INTO THE CONTAINER VIA SSH

Assuming that there is no firewall between the host that runs the container and the remote, here is how the user would access the container:

```
[remote]$ ssh root@<host name> -p <alternate port number>
[container]# source $VERA_INSTALL_DIR/load_env.sh
```

The user gives his SSH public key. In exchange, he receives the name of the host and the port number.

4. EVALUATING AMAZON GOVCLOUD FOR VERA APPLICATIONS

The Amazon Web Services (AWS) Elastic Compute Cloud (EC2) service allows for users to allocate on-demand computing resources which can be scaled to demand. New hosts ("instances" in AWS parlance) can be spawned as needed for a single calculation, then released when the calculation or series of calculations has been completed. This is a potentially attractive option for CASL industry partners that do not have the resources to purchase and maintain their own computing infrastructure to perform large VERA calculations. GovCloud is a subset of the AWS infrastructure which satisfies the necessary security requirements to work with sensitive and export-controlled code, making it theoretically possible to run VERA in this environment.

While there is no doubt that VERA can run in this environment, it is not clear how well the EC2 platform will perform under the VERA applications. The EC2 platform is primarily designed for scalable web services and map-reduce operations, which tend not to need large memory bandwidth between hosts. This is in contrast to traditional HPC clusters, which feature high-speed interconnects between hosts for maximum possible memory bandwidth. Amazon boast that their compute optimized instance types have network performance of about 10 Gigabit/s, while modern Infiniband interconnects features tens to hundreds of Gigabit/s. This large difference in host-to-host communication behavior may have a large impact on distributed-memory scaling performance. Furthermore, EC2 instances by default operate on virtualized hardware, which may map to physical hardware shared by another instance. Depending on the specific circumstances, this may lead to erratic performance. For a cost premium, AWS users may request exclusive access to hardware, or even dedicated hosts for their instances; this may be necessary to realize decent performance, but should be investigated.

Several CASL members have obtained accounts on the AWS GovCloud and begun experimenting with the platform. Since we are still waiting for Site Office approval to deploy export-controlled software to the GovCloud, it was decided that COBRA-TF would be used for initial evaluation. While a Docker-based approach may be preferable in the long run, for simple experimentation with setting up a collection of EC2 instances and driving them as an MPI cluster, the stock Amazon Machine Image (AMI) format was used directly to encapsulate the software environment used for testing. The AMI image contains the TPLs necessary to build COBRA-TF, the COBRA-TF git repository, and a script to simplify configuring the COBRA-TF build system:

```
#!/bin/bash

cmake -DCMAKE_BUILD_TYPE:STRING=Release \
      -DTPL_ENABLE_MPI:BOOL=ON \
      -DTPL_ENABLE_HDF5:BOOL=ON \
      -DHDF5_LIBRARY_DIRS=/opt/hdf5/lib \
      -DHDF5_INCLUDE_DIRS=/opt/hdf5/include \
      -DHDF5_LIBRARY_NAMES="hdf5;hdf5_fortran" \
      -DTPL_ENABLE_PETSC:BOOL=ON \
      -DPETSC_LIBRARY_DIRS=/opt/petsc/opt/lib \
      -DPETSC_INCLUDE_DIRS=/opt/petsc/opt/include \
      ../
```

The image also contains a simple bash script that automates the process of setting up all of the EC2 instances as an MPI cluster. For such a cluster to function, all nodes must interact with a unified NFS filesysem, and a manifest of the all nodes in the cluster must be stored in a “hostfile”. At the time of this writing, the IP addresses of all of the nodes must be manually entered in the “node” file which is taken as input to the script, though in the future this task may also be automated.

```
#!/bin/bash

# Get master IP address
IPADDR=$(ifconfig eth0 | grep 'inet addr:'| grep -v '127.0.0.1' | cut -d: -f2 | awk '{ print $1}')
NPROC=$(grep -c ^processor /proc/cpuinfo)
echo Master IP: $IPADDR
echo With $NPROC cores

# Setup NFS to share home directory with nodes
sudo mkdir -p /exports/home
sudo mount --bind /home /exports/home
sudo echo "/exports␣10.0.0.0/24(rw,fsid=root,no_subtree_check)" > /etc/exports
sudo echo "/exports/home␣10.0.0.0/24(rw,no_subtree_check,nohide)" >> /etc/exports
sudo exportfs -rav
sudo service nfs restart

# Start the hostfile
echo $IPADDR:8 > hostfile

# configure nodes. A list of all addresses for all nodes other than the master
# should be in a file called 'nodes'.
# Each entry will be:
#  - added to the hostfile with the appropriate number of processors specified
#  - instructed to mount the master's NFS share to their /home
readarray NODES < nodes
echo Nodes: ${NODES[0]} ${NODES[1]}
for p in "${NODES[@]}"
do
    echo Configuring node at: $p
    ssh $p hostname
    NPROC=$(ssh $p grep -c ^processor /proc/cpuinfo)
    echo Has $NPROC processors
    echo $p:$NPROC >> hostfile
    ssh -t $p bash -c "'
␣␣␣␣␣␣␣␣cd␣/
␣␣␣␣␣␣␣␣sudo␣mount␣$IPADDR:/exports/home␣/home
␣␣␣␣'"
done
```

Following initialization of the cluster, parallel jobs may be started with a command similar to any other MPI cluster:

```
mpirun -np [# processors] --hostfile ~/hostfile [executable]
```

It remains as future work to perform detailed scaling studies using VERA components to determine if the AWS GovCloud system is suitable for large scale VERA calculations.

5. BINARIES ON INSPECT

The delivery mechanism was developed to build the InSPECT binaries from source on remus.ornl.gov, since we do not allow source code on to be placed on inspect.ornl.gov. The operating systems and directory structure are consistent between remus and inspect, which allows us to build on one machine and be binary compatible on the other, even with the large number of TPLs and MPICH libraries that are linked to the VERA executables. Also it was necessary to place all third party and MPICH libraries on inspect which were also built on remus.ornl.gov. Prior to release of VERA 3.6, RSICC received a general release of the ORNL office of export control on March 15th and processing the package to make available for selected users.